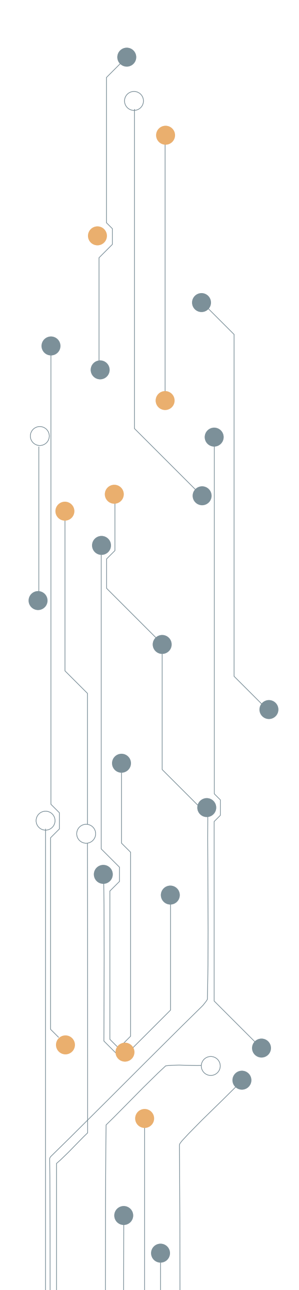
Diseño de algoritmos



Índice



Diseño de algoritmos

1. Diseño de algoritmos

La fase de diseño de un programa resulta es sin duda alguna una de las más importante de todo el proceso de creación de un programa. En ella se diseña el algoritmo que describe lo que tiene que hacer el programa. Esta descripción se realiza siguiendo la lógica de programación

Por lógica de programación entendemos la capacidad para expresar la secuencia de operaciones o algoritmo que debe seguir un programa informático para resolver un problema, utilizando para ello una serie de estructuras o expresiones estandarizadas.

La lógica de programación no es algo con lo que se nazca, sino que se adquiere con el tiempo y con la práctica. Por ello, el objetivo de este capítulo es presentarte todos los elementos necesarios para que, dadas unas especificaciones, seas capaz de expresar de forma adecuada la lógica de operaciones que un programa debería seguir para conseguir su objetivo.

Según hemos indicado en el capítulo anterior, dos son las técnicas que se utilizan para diseñar el algoritmo de un programa:

- Diagramas de flujo
- Pseudocódigo

A continuación analizaremos dichas técnicas con detalle y veremos numerosos ejemplos a fin de que adquieras la soltura necesaria para diseñar tus propios programas.

Pero antes de entrar en las técnicas de diseño de algoritmos, vamos a analizar uno de los elementos claves de la programación y, por tanto, del diseño de algoritmos: los datos.

1.1 | Datos y variables

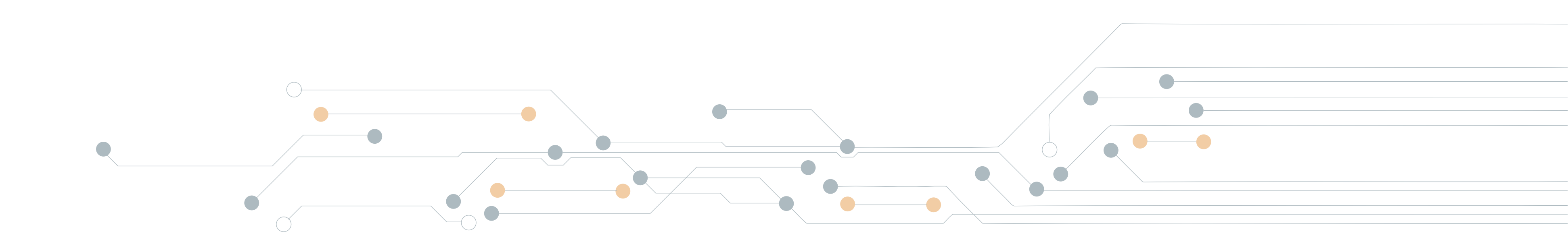
Durante el desarrollo de un programa informático, independientemente del lenguaje de programación utilizado, se utilizan datos y variables. Los datos representan la información manejada por el programa, y pueden ser números, fechas, textos etc. Todo programa recibe unos datos de entrada que son manipulados por el mismo y, como resultado, generará unos datos de salida.

Durante la manipulación de los datos por parte del programa, estos son almacenados en variables. Una variable es una zona de memoria, a la que se le asigna un nombre o identificador, en la que se guarda un dato de un determinado tipo. Después, la variable puede utilizarse dentro del programa para realizar diferentes operaciones con el dato que contine.

Normalmente, los lenguajes de programación obligan a declarar las variables antes de ser utilizadas. Declarar una variable es la manera de indicarle al compilador que queremos utilizar una variable, a fin de que éste pueda reservar el espacio en memoria para el almacenamiento de la misma.

La manera en la que un lenguaje declara una variable depende de la sintaxis del mismo. El siguiente ejemplo corresponde a una declaración de una variable de tipo entero en Java:

```
int num;
```

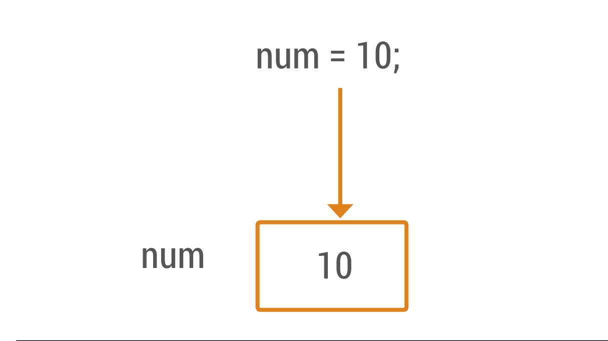


En este caso, se declara una variable de tipo entero, es decir, que solo puede almacenar datos numéricos de tipo entero, y se le asigna el identificador num. Gráficamente, la situación en memoria podría representarse así:

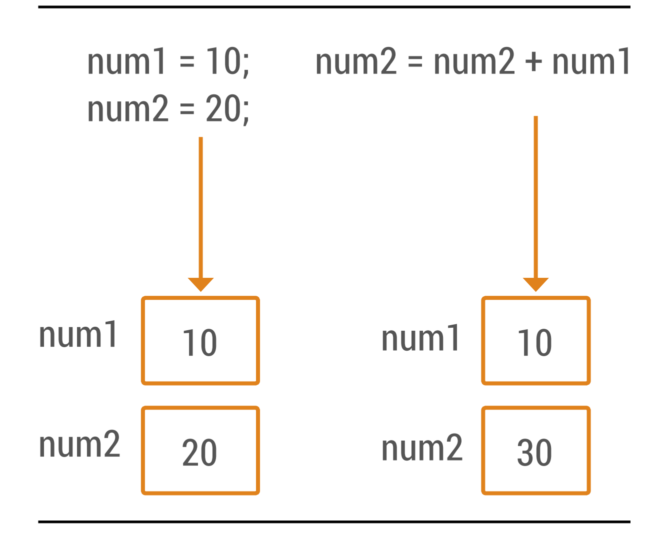
num

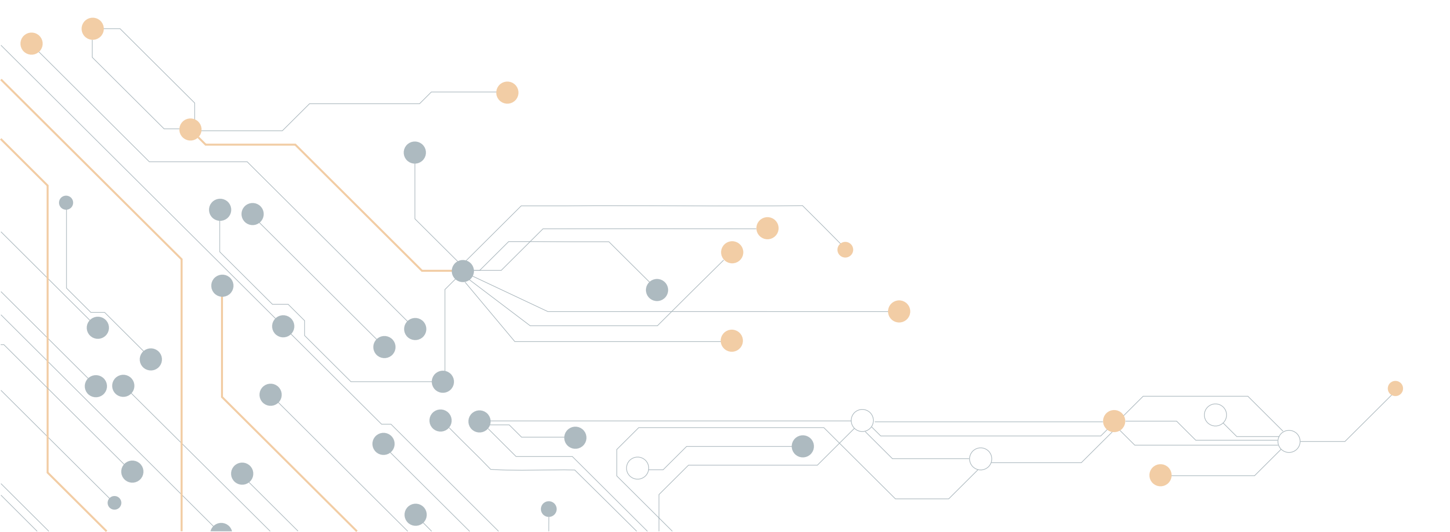
La caja sería como la zona de memoria asignada a la variable y el nombre que aparece a su izquierda sería el identificador asignado a la misma.

Una vez declarada la variable, el programa puede hacer uso de ella para asignarle datos y después realizar operaciones con los mismos. En el ejemplo anterior, la asignación de un dato a la variable se realizaría a través del signo =, tal y como se muestra en el ejemplo:



Como decimos, a partir del momento en que la variable ya tiene el dato, pueden realizarse operaciones con el mismo, incluido la modificación del valor de la propia variable. El siguiente ejemplo, sumaría el contenido de dos variables y lo depositaría en una de ellas:



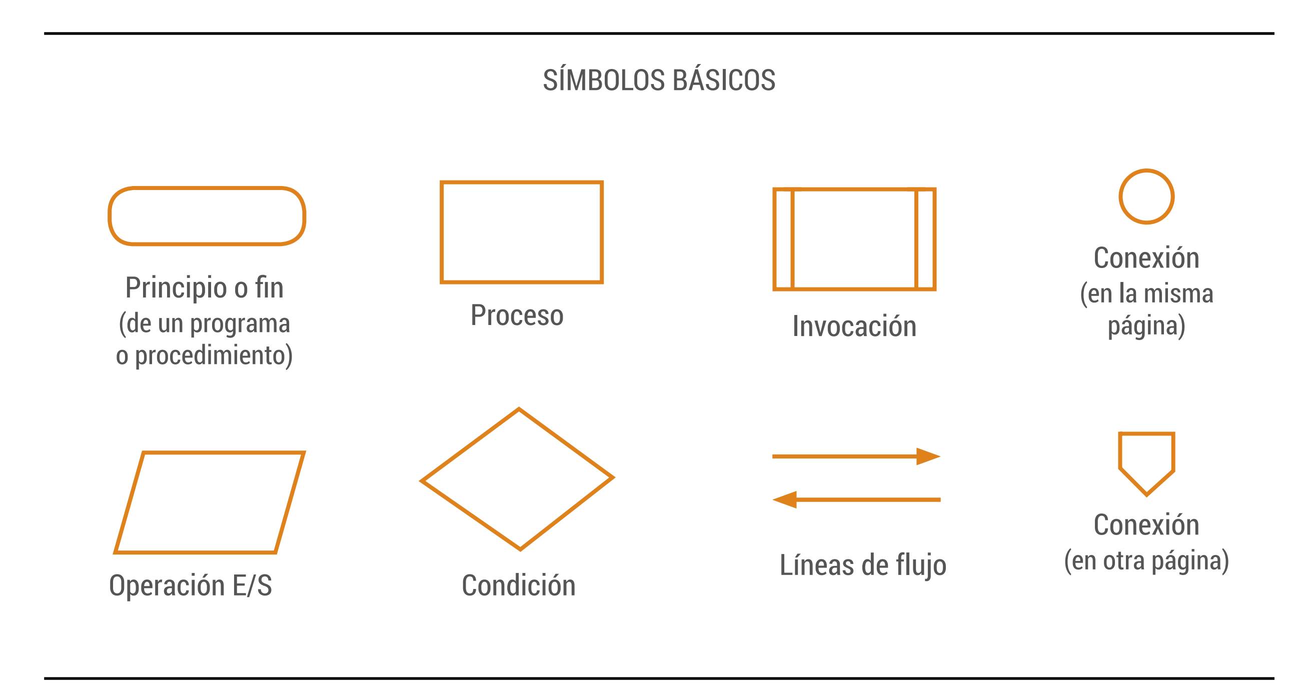


1.2 | Diagramas de flujo

Conocidos también como **ordinogramas**, los diagramas de flujo son una técnica que se hizo muy popular en los principios de la programación de ordenadores. Aunque actualmente prácticamente no se utilice, resulta muy útil para ir adquiriendo esa lógica de programación que nos permita diseñar y desarrollar programas, pues el uso de elementos gráficos para representar las estructuras típicas utilizadas en programación facilitará la comprensión de los mismos.

Los ordinogramas se emplean durante la fase de diseño de un programa y son independientes del lenguaje de programación utilizado para implementar el mismo.

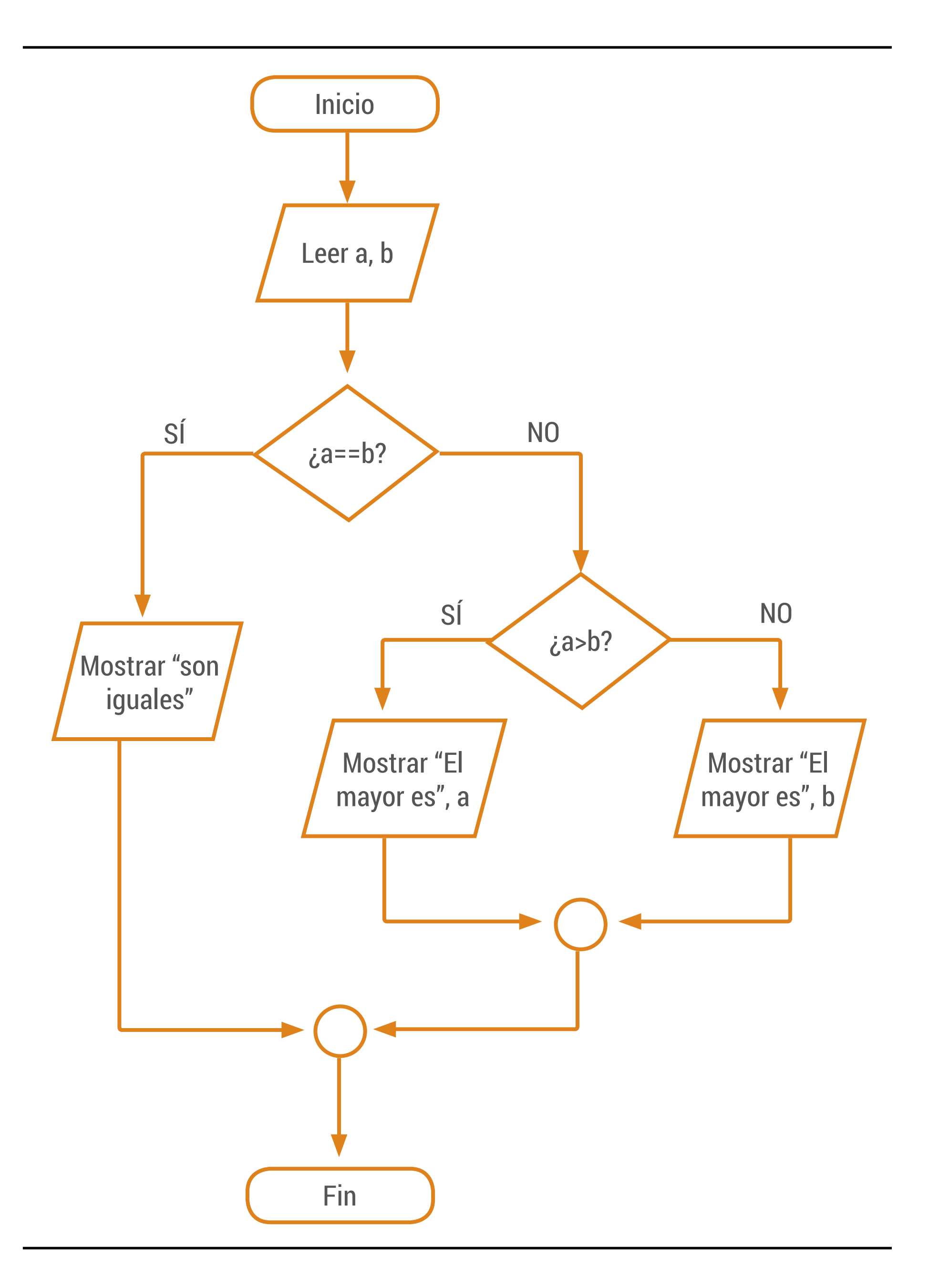
Para crear un ordinograma se utilizan una serie de símbolos bastante intuitivos y fáciles de recordar. La siguiente imagen nos muestra los símbolos básicos utilizados en la creación de un diagrama de flujo:



A la hora de utilizar estos símbolos para crear un ordinograma hay que tener en cuenta ciertas normas básicas que debemos cumplir:

- Todos los símbolos utilizados deben estar unidos por líneas de flujo solo horizontales y/o verticales.
- No se pueden cruzar las líneas de flujo. Evitar los cruces.
- No deben quedar líneas de flujo sin conectar.
- A un símbolo de proceso pueden llegarle varias líneas de flujo pero solo puede salir una de él.
- Al símbolo de inicio no puede llegarle ninguna línea de flujo.
- De un símbolo de fin no puede salir ninguna línea de flujo pero si le pueden llegar varias.
- Se deben trazar los símbolos de manera que se pueda leer de arriba abajo y de izquierda a derecha.

El siguiente ordinograma de ejemplo corresponde al algoritmo de un programa encargado de leer dos números y mostrar sin son iguales o, en caso de ser distintos, cuál de ellos es el mayor.



1.3 | Símbolo de principio y fin

Como su nombre indica, se utilizan para especificar el principio y fin del programa. En su interior se escribe la palabra inicio o fin, según corresponda.

Inicio   Fin

1.4 | Símbolo de proceso

El símbolo de proceso especifica una operación realizada por el programa, como por ejemplo, una operación aritmética con los datos recibidos en la entrada. La operación a realizar se especifica en el interior del rectángulo:

s = a + b

EJEMPLO OPERACIÓN

En el ejemplo anterior, se realiza la suma de dos datos representados por las variables a y b y el resultado se deposita en una tercera variable llamada “s”. Es decir, la asignación se realiza de derecha a izquierda, que es como se hace en la mayoría de los lenguajes de alto nivel, utilizando el signo “=”.

Para expresar las operaciones empleamos símbolos, como el signo + para la suma, el * para la multiplicación, el = para asignar un valor, etc. Estos símbolos son conocidos como **operadores.**

Cada lenguaje de programación tiene su propio juego de operadores, por lo que a la hora de utilizarlos en el diseño de un ordinograma no hay un convenio claro sobre su utilización; se debe procurar que sean lo más intuitivos posibles. Durante el estudio de la programación estructurada estableceremos un juego de operadores para ser utilizados durante el diseño de pseudocódigo.

1.5 | Líneas de flujo

Las líneas de flujo representan el camino que sigue el algoritmo. Para representarlas se utilizan flechas, que parten de una operación y apuntan a la siguiente operación a realizar.

1.6 | Símbolo de entrada /salida

Especifican una operación de entrada o salida, es decir, de entrada de datos al programa o de salida de datos desde el programa al exterior. La operación en concreto se indica en el interior del símbolo, utilizando un verbo como "leer", "mostrar", "imprimir", etc.

Leer a, b

Mostrar "El mayor es", a

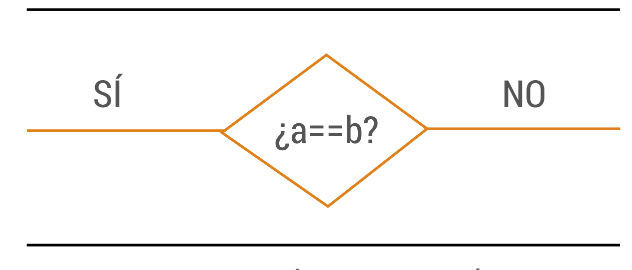
EJEMPLO SÍMBOLO DE ENTRADA / SALIDA

Al indicar la operación de entrada o salida en el diagrama de flujo, nos debemos abstraer del dispositivo en concreto que se utilizará para la operación. Por ejemplo, al indicar Leer a, b, simplemente indicamos que el programa recibe dos datos del exterior, sin especificar el dispositivo concreto utilizado para la lectura de los mismos (teclado, fichero, Web, etc.).

En el caso de las operaciones de entrada, a continuación del nombre de la operación (leer en este caso) se indican las variables donde se guardarán los datos recuperados, separadas por una coma. En el caso de las operaciones de salida, a continuación del nombre de la operación se indicará la lista de variables con los datos a mostrar y si queremos que aparezca también una frase como en nuestro ejemplo, esta frase se escribirá entre comillas.

1.7 | Símbolo de decisión

Mediante este símbolo con forma de rombo expresamos una operación de comprobación que puede alterar el flujo de ejecución de un programa. El resultado de la comprobación será de tipo verdadero/falso (SI/NO), de modo que si es verdadero el programa tomará un camino y si es falso tomará otro:

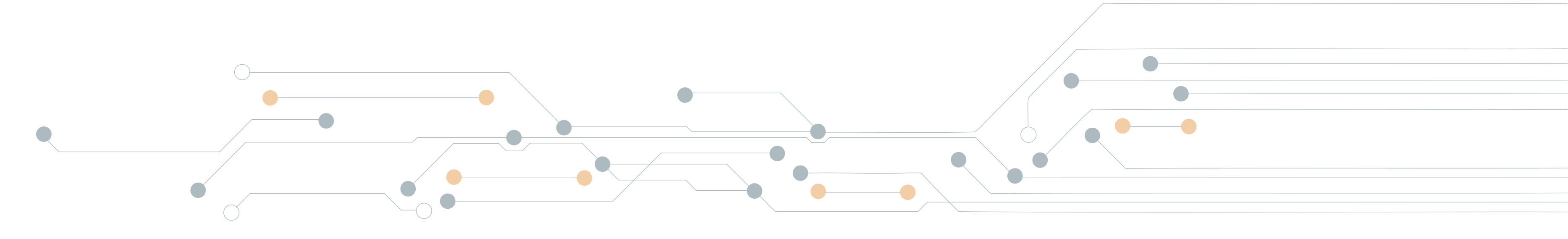


EJEMPLO SÍMBOLO DECISIÓN

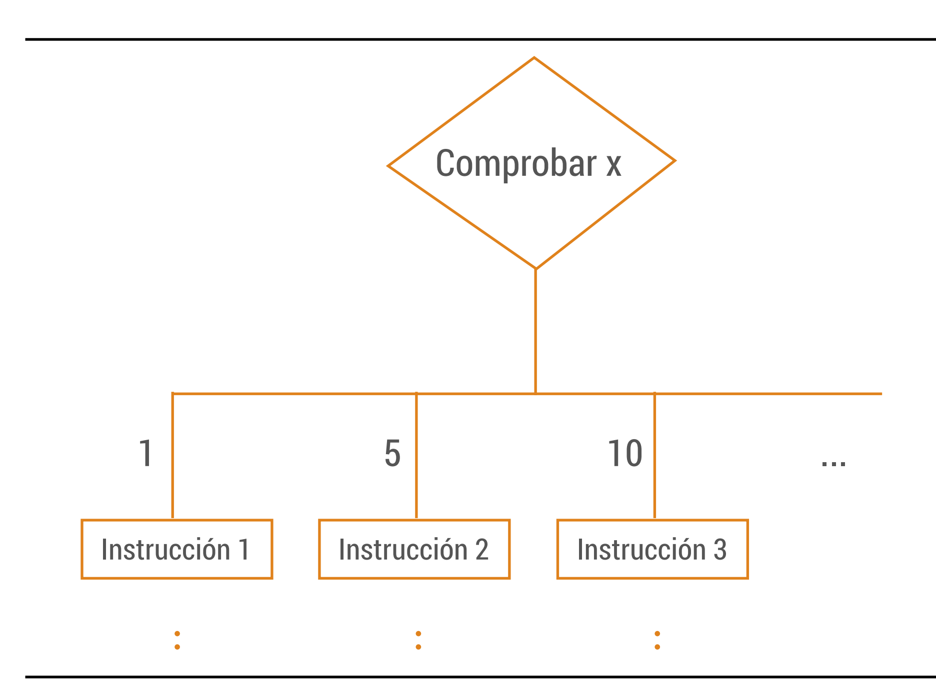
La operación de comprobación se indica entre interrogaciones en el interior del rombo y se utilizan los llamados operadores de tipo condicional, es decir, aquellos símbolos que se utilizan para comprobar la igualdad o desigualdad de datos, si un dato es mayor o menor que otro, etc. El resultado de estas comprobaciones siempre será verdadero o falso.

En el ejemplo que hemos presentado, se pregunta si el dato contenido en a es igual al contenido en b, para lo que usamos el operador doble igual ==, en vez del simple igual =, pues éste se reserva para operaciones de asignación.

Una variante de este tipo de operaciones son las decisiones con salida múltiple o alternativas múltiples. En este caso, la operación de comprobación puede dar como resultado distintos valores, no verdadero o falso, por lo que se podrán definir tantas salidas o caminos como posibles resultados queramos controlar.



La forma de expresar una operación de decisión múltiple sería como se indica en el ejemplo:

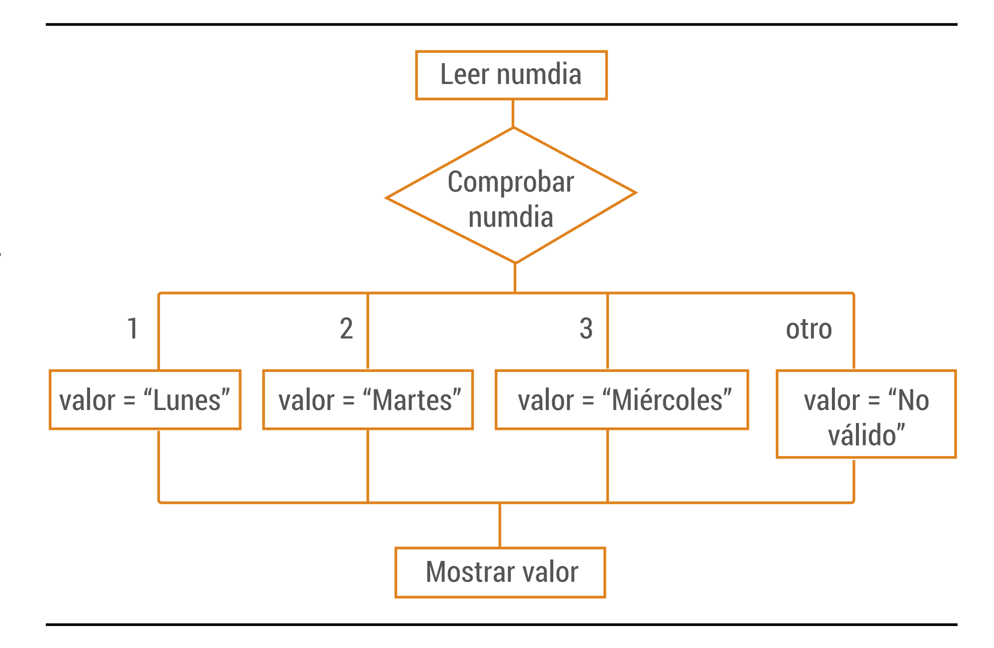


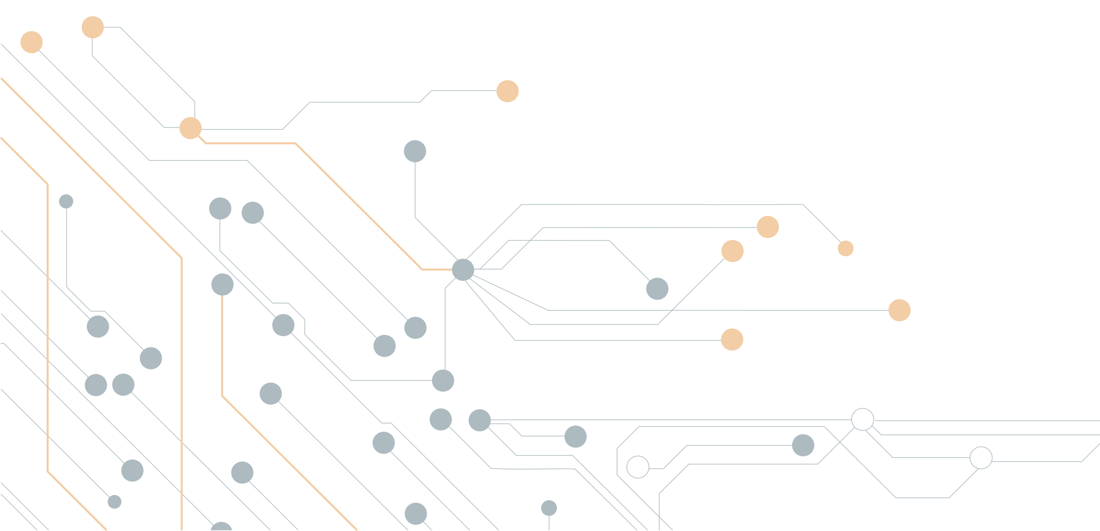
EJEMPLO OPERACIÓN DECISIÓN MÚLTIPLE

Como vemos en el ejemplo, dependiendo del valor de un dato llamado "x", el programa puede tomar diferentes caminos y ejecutar diferentes grupos de instrucciones; si es 1, ejecutará las instrucciones del grupo "instrucción 1", si es 5 las del grupo "Instrucción 2", etc. Se puede definir también un camino para el caso de que el resultado de la expresión no coincida con ninguno de los valores definidos.

Después de ejecutar cada caso, el programa continuaría por un único camino.

El siguiente ordinograma de ejemplo se encarga de leer un número de día y mostrar el día correspondiente, siempre que sea un número entre 1 y 3, sino mostrará que es un día no válido:





1.8 | Acumuladores y contadores

En la resolución de algoritmos nos vamos a encontrar numerosas situaciones en las que tenemos que realizar la suma o producto de varias cantidades. Estas operaciones no se realizarán en una única fase, sino que requerirán la realización de sumas o productos parciales de forma repetitiva hasta conseguir la cantidad final.

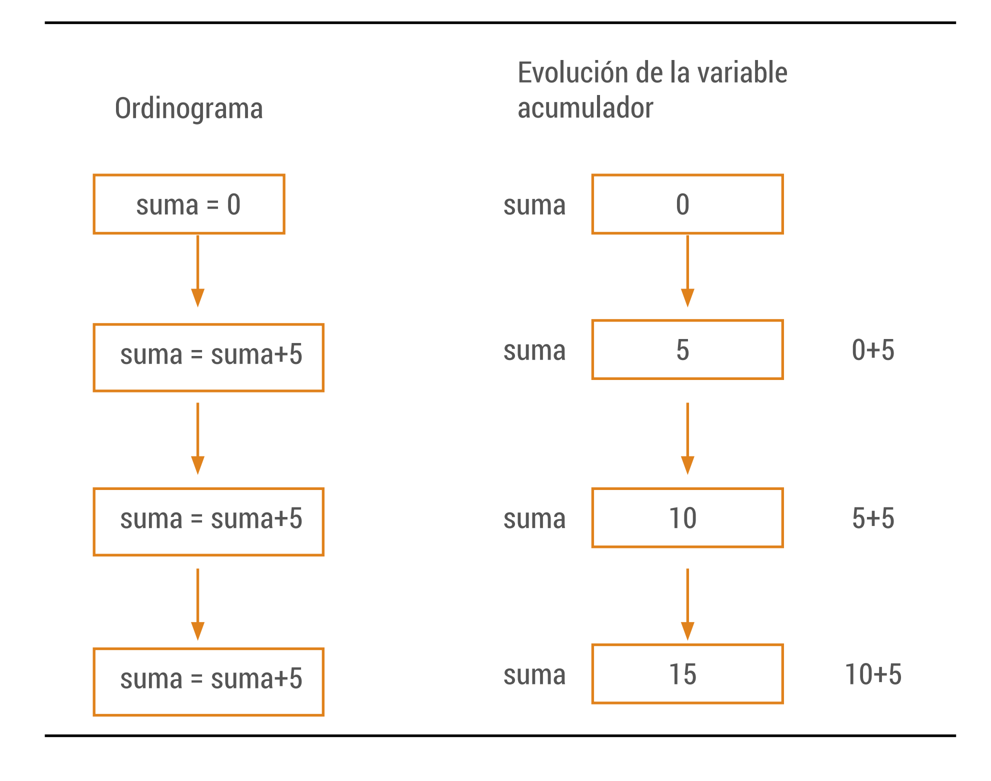
Para guardar los resultados de estas operaciones utilizaremos variables de tipo numérico, a las que se conoce en programación con el nombre de acumuladores, puesto que van acumulando los resultados parciales de la operación.

En cada suma parcial que se realiza con un acumulador se debe tomar el valor de la variable, sumarle o multiplicarle la cantidad y depositar el resultado de nuevo en la variable. Esto se representa en un ordinograma mediante el símbolo de proceso utilizado una expresión del tipo:

```
variable=variable+cantidad o variable=variable*cantidad
```

En el caso de los acumuladores de suma, la variable se inicializará a 0 mientras que para la multiplicación lo hará a 1.

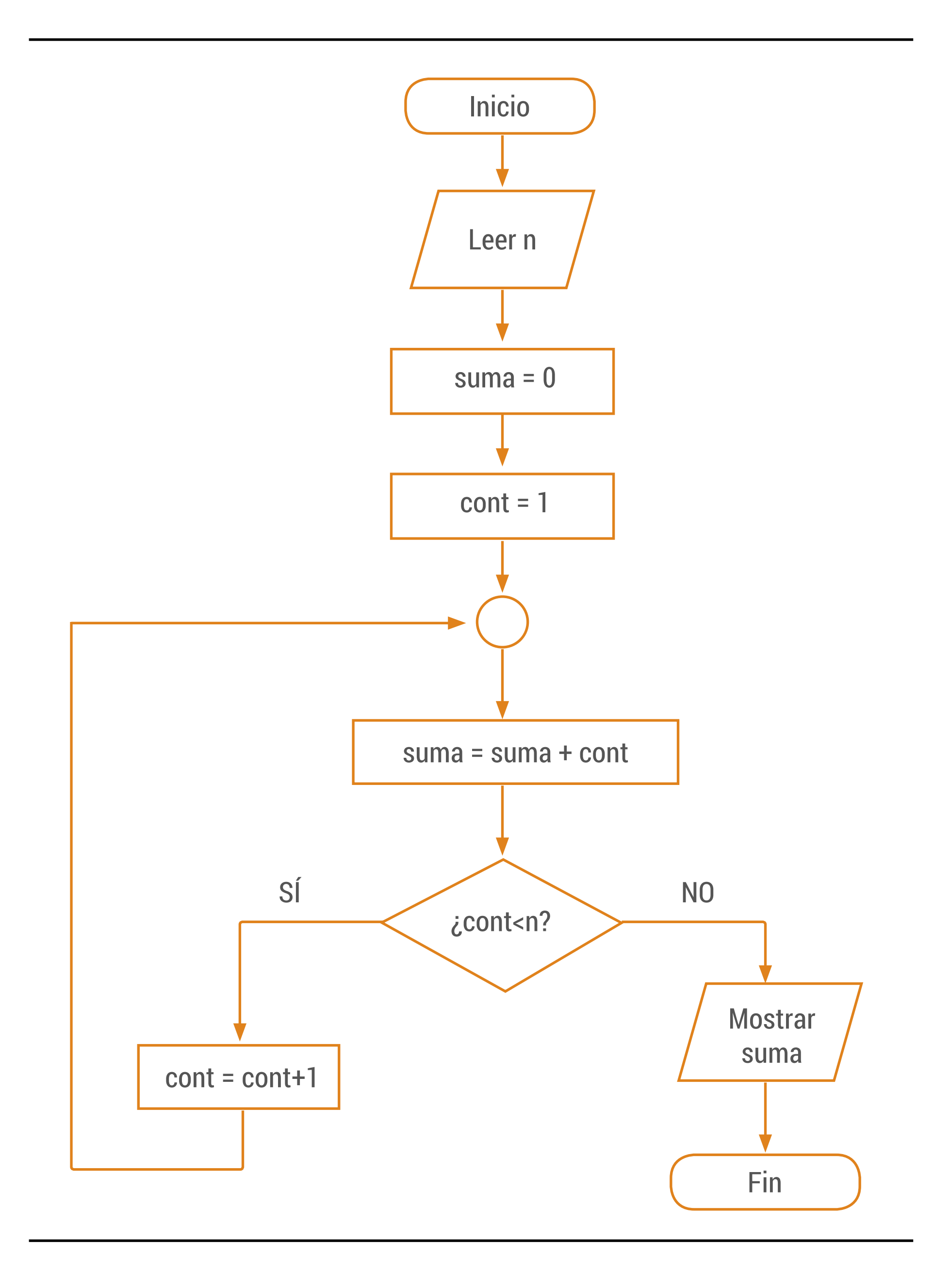
En el siguiente ejemplo se muestra el caso de un pequeño programa que realiza la suma de los tres primeros múltiplos de 5 utilizando un acumulador



Otro tipo de variables numéricas comúnmente utilizadas en un programa informático son los **contadores**. Un contador es una variable que se va incrementando en una unidad cada vez que el programa realiza una determinada acción, a fin de ir contando el número de veces que dicha acción es realizada. También puede haber decontadores, es decir, variables que vayan restando uno a su valor repetidas veces.

La operación de incremento se representaría en un organigrama mediante el símbolo de proceso en el que se indicará una expresión del tipo variable=variable+1

A continuación, te presentamos otro ejemplo de ordinograma que resuelve otro sencillo algoritmo, concretamente, se trataría de un programa que calcula la suma de todos los números naturales comprendidos entre 1 y un número leído. Por ejemplo, si el número leído por el programa es 8, el resultado sería 1+2+3+4+5+6+7+8. Este es su diagrama de flujo:



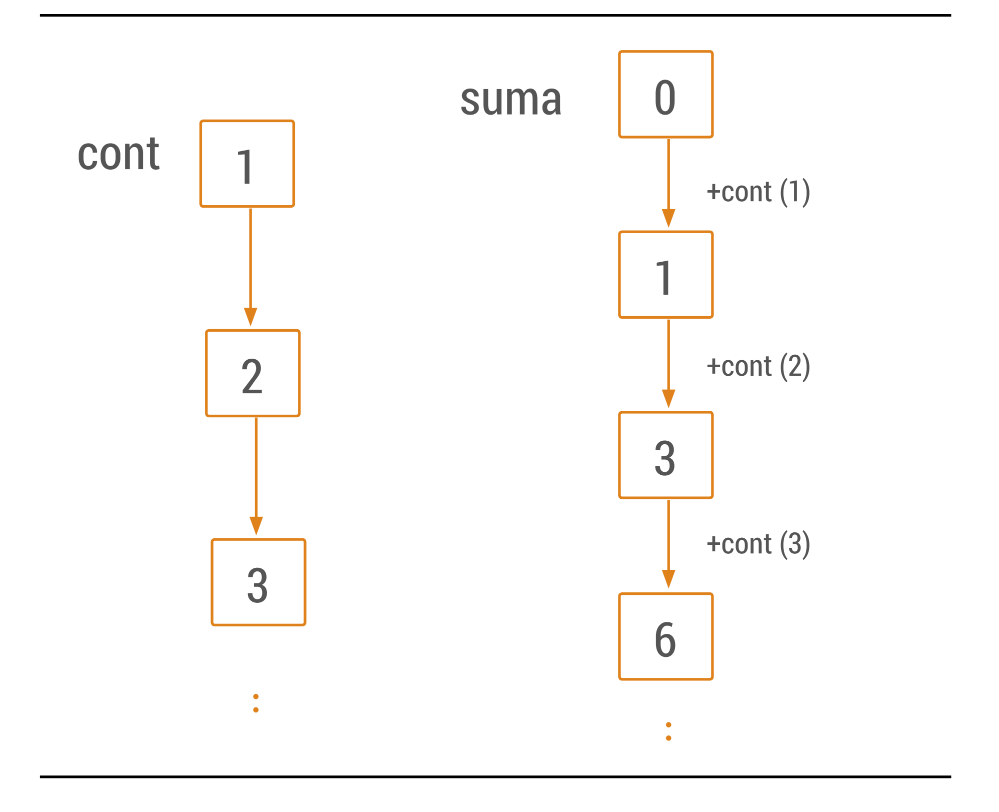
Como vemos, a parte de la variable *n* empleada para guardar el número leído, se utiliza una variable acumulador llamada *suma* y otra de tipo contador llamada cont.

En *suma* iremos acumulando los valores de las sumas parciales realizadas, mientras que *cont* irá contando los números desde 1 a n para ir sumándolos después con el contenido de suma mediante la instrucción *suma=suma+cont.*

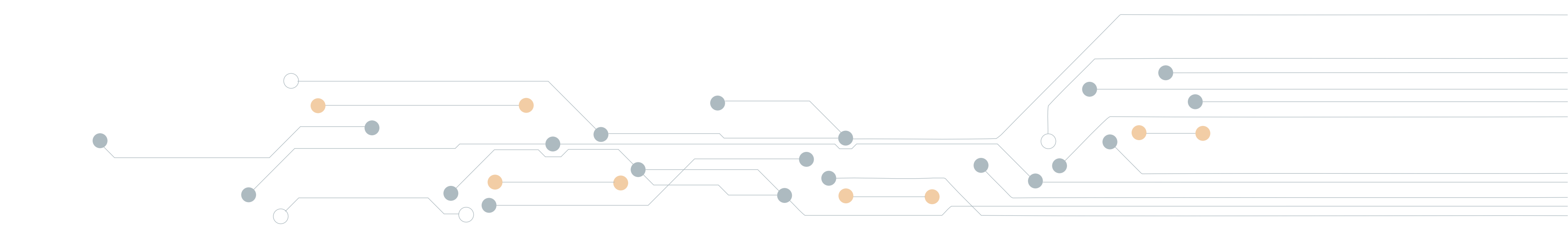
Así, cuando el programa ejecuta por primera vez la operación anterior, como *suma* se ha iniciado a 0 y cont a 1, la operación sería *suma*=0+1, con lo que se guardaría el valor 1. Después, el programa pregunta si *cont<n*, como *cont* vale 1, el resultado de la pregunta es verdadero (suponemos que n es positivo y mayor que 1), por lo que se incrementa el valor de cont en una unidad (ahora valdrá 2) y se vuelve a la instrucción *suma=suma+cont.*

Al ejecutar por segunda vez la instrucción, tendríamos *suma*=1+2 y de nuevo se volvería a realizar la consulta *cont<n*, así hasta que *cont* haya alcanzado a *n,* en cuyo caso ya se habrán acumulado en *suma* todos los números comprendidos entre 1 y *n.*

En el siguiente diagrama se ilustra la evolución de ambas variables en el programa:



Los contadores y acumuladores siempre se utilizan en el interior de estructuras repetitivas, es decir, aquellas que contienen bloques de instrucciones que se ejecutan repetidamente.

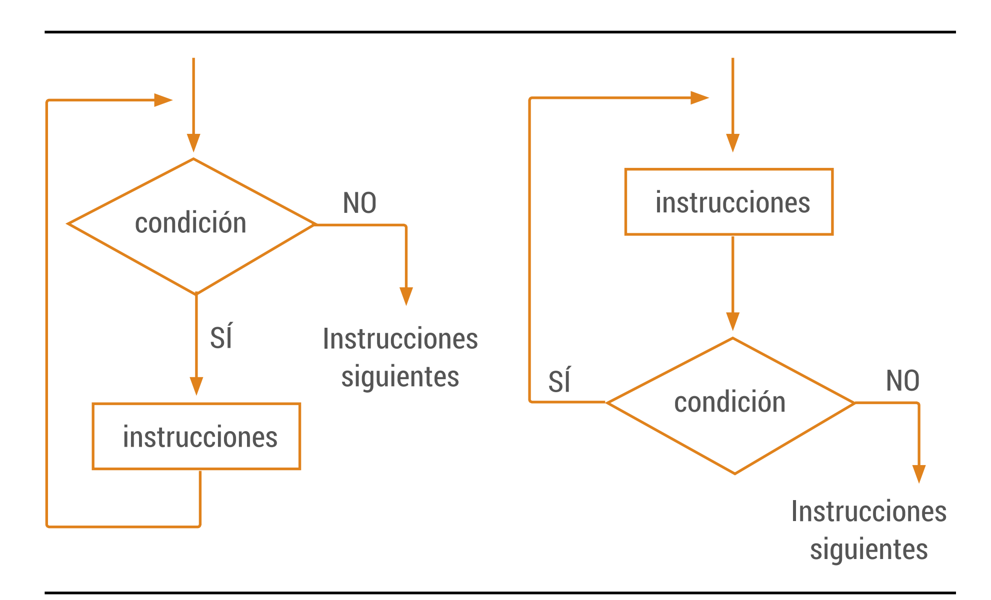


1.9 | Estructuras repetitivas

En el programa de ejemplo anterior hemos utilizado una estructura clásica en programación, que es la **estructura repetitiva**. No se trata de una operación en sí, sino de un conjunto de operaciones que se ejecutan mientras que se de una condición.

En nuestro caso, se trata de ir acumulando la suma de números naturales en la variable suma hasta alcanzar el número leído.

En una estructura repetitiva siempre encontramos una instrucción de tipo condicional, que determina si el grupo de instrucciones tiene que volver o no a ejecutarse. Se puede optar por comprobar la condición y ejecutar las instrucciones si se cumple, o ejecutar primero las instrucciones y comprobar después la condición:



En cualquiera de los casos, si la condición se cumple (resultado verdadero), el bloque de instrucciones vuelve a ejecutarse y en el momento en que deje de ejecutarse se continuará por otro camino.


Telefonica
EDUCACIÓN DIGITAL